UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/622,022	07/17/2003	John B. Bley	WILY-01016US0	1684

28554 7590 01/16/2007
VIERRA MAGEN MARCUS & DENIRO LLP
575 MARKET STREET SUITE 2500
SAN FRANCISCO, CA 94105

EXAMINER	
CHOW, CHIH CHING	
ART UNIT	PAPER NUMBER
2191	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/16/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	**Application No.** 10/622,022	**Applicant(s)** BLEY ET AL.	
	Examiner Chih-Ching Chow	**Art Unit** 2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 July 2003*.

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-47* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-47* is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *17 July 2003* is/are: a)☒ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)□ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____.

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) □ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) □ Notice of Informal Patent Application
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *12/2/05*.	6) □ Other: _____.

DETAILED ACTION

1. This action is responsive to the application filed on July 17, 2003.

2. The priority date considered for this application is July 17, 2003.

3. Claims 1-47 have been examined.

Specification

4. The use of the trademark JAVA has been noted in this application. It should be capitalized wherever it appears and be accompanied by the generic terminology.

Although the use of trademarks is permissible in patent applications, the proprietary nature of the marks should be respected and every effort made to prevent their use in any manner which might adversely affect their validity as trademarks. To expedite correction on this matter, the examiner suggests the following guidelines for Applicant to follow in amending the specification:

a. Capitalize each letter of a trademark or accompany the trademark with an appropriate designation symbol, e.g., ™ or ®, as appropriate.

b. Use each trademark as an adjective modifying a descriptive noun. For example, it would be appropriate to recite "a JAVA class file" or "a JAVA application". Note that in these examples, "class file" and "application" provide accompanying generic terminology, describing the context in which the trademark is used. By itself, the trademark JAVA specifies only the source of the so-labeled products, namely SUN Microsystems, Inc.

Claim Rejections - 35 USC § 112

5. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

6. Claim 1 recites the limitation "said additional method". There is insufficient antecedent basis for this limitation in the claim.

Claim Rejections – 35 USC § 101

7. 35 U.S.C. § 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the condition and requirements of this title.

8. Claims 12-17, 25-26 are rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. For example, claim 12 recites various steps of storing result from an operand stack (a data structure), and preparing operand stack, invoking and finally resetting the operand stack back to the state prior to the original storing step; which does not achieve the final result (accessing information), the claims do not have a practical application and do not produce a useful, concrete, and tangible result.

Claim Rejections - 35 USC § 102

9. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

10. Claims 1-7, 9-19, 21-26, 29-32, 34-38, 41-43, 45-47 are rejected under 35 U.S.C. 102(b) as being anticipated by US Patent No. 6,289,446, by Nilsson, hereinafter "Nilsson".

As Per claim 1, Nilsson discloses:

- ***A method for adding functionality in order to access information, comprising: accessing existing object code, said existing object code includes a first method,***

said first method is capable of providing a result; and adding new code to said first method, said new code provides said result to said additional method. Nilsson's disclosure teaches a method to access existing object code, see Nilsson's Abstract, "In-code context data used for **exception handling** is incorporated into a **special call instruction** which is recognized by the processor. The information is skipped at the time of the function call and read at the time of the stack unwinding." And column 1, "Linking may be thought of as the general process of combining or linking together one or more **compiled object modules** to create an executable program. This task usually falls to a program called a 'linker.' In typical operation, a linker receives, either from the user or from an integrated compiler, a list of **object modules** desired to be included in the link operation. The linker scans the **object modules** from the **object and library files specified**. After resolving interconnecting references as needed, the linker constructs an executable image **by organizing the object code** from the modules of the program in a format understood by the operating system program loader. (*accessing existing object code*) The end **result of linking is executable code** (typically an .exe file) which, after testing and quality assurance ***(capable of providing a result)***, is passed to the user with appropriate **installation and usage instructions**, or to a factory **for installation** in products with embedded computer systems"; further column 6, lines 45-55, "According to the present invention, the **in-code context data is incorporated into a special call instruction** which is recognized by the processor (*the linking of the special call instruction causes adding new code to said first method*). The information is skipped at the time of the function call and read at the time of the stack unwinding. This **special call instruction may be implemented to run** (*adding new code to said first method, said new code provides said result to said additional method*) at no extra cycle costs compared to normal instructions, except for the external execution time dependencies from such machinery as a cache involved in the instruction fetching, since it would never be necessary during normal execution to **actually access the information. The information is only accessed during exception handling**. *(adding functionality in order to access information)*"

As Per claim 2, Nilsson discloses:

- ***The method of claim 1, wherein the validation actions comprise a validation program associated with the software application that, when executed, returns results indicating whether aspects of the software application are properly installed on the target computer.***

For claim 1 feature see claim 1 rejection, for 'validation action' see Nilsson's column 2, lines 1-3, "The end result of **linking is executable code** (typically an .exe file) which, **after testing and quality assurance**, (*validated) is* passed to the user with appropriate installation and usage instructions".

As Per claim 3, Nilsson discloses:

- ***A method according to claim 1, wherein: said result includes a reference to an exception.***

For claim 1 feature see claim 1 rejection, for rest of claim 3 feature see claim 1 rejection, where '**the information is only accessed during exception handling**'.

As Per claim 4, Nilsson discloses:

- ***A method according to claim 1, wherein said step of adding new code includes: adding code that stores said result for said first method from an operand stack; adding code that prepares said operand stack for an invocation of said additional method; adding code that invokes said additional method, including providing said result to said additional method; and adding code that***

resets said operand stack with respect to said result to a state existing prior to storing said result.

For claim 1 feature see claim 1 rejection, for rest of claim 4 feature see Nilsson's column 3, lines 9-25, "The essence of a function call is that it must pass **any arguments (or parameters) to the target function** (*operand stack)*, transfer control to the memory section holding the function's executable code, **return the result of the call,** and at the same time, **store sufficient information to ensure that subsequent execution resumes immediately after the point** where the original **function call** was made *(adding code that stores said result for said first method from an operand stack; adding code that prepares said operand stack for an invocation of said additional method)*. This function-calling mechanism, is usually achieved by pushing and **pulling data and memory addresses on and off a stack** (*from an operand stack*) prior to, during, and after, the call. A **stack** is simply a dedicated portion of memory usually organized as a LIFO (last in, first out) data structure. The **stack** is not normally manipulated directly by the programmer, but its **contents are changed** as **a result of** the **function calls coded by the programmer** (*adding code that resets said operand stack with respect to said result to a state existing prior to storing said result)*. Programs do have **direct access** to another portion of memory, often called the heap, and a key element in **exception handling** involves the management of this vital resource." And see Nilsson's column 8, lines 23-33, "The method includes **fetching a sequence of instructions** from the memory in response to an instruction pointer identifying an address in the addressable memory. Next, a current instruction is decoded including detecting the context-call-instruction. The instruction pointer is updated by the normal length of the context-call-instruction plus the determinant length of the context data in response to detection of the context-call-instruction, else the instruction pointer is updated by the length of the current instruction, **which includes any operands** if appropriate. *(operand stack for an invocation)*".

As Per claim 5, Nilsson discloses:

- ***A method according to claim 4, wherein said step of adding new code further includes: adding code that returns said result after resetting said operand stack, said result is a return value.***

See claim 4 rejection *(return the result of the call).*

As Per claim 6, Nilsson discloses:

- ***A method according to claim 4, wherein: said result includes an exception; and said step of adding new code further includes adding code that throws said exception after said step of resetting, said result represents an exception.***

See claim 4 rejection, for rest of claim 6 feature see Nilsson's column 3, lines 44-50, "**exceptional conditions and errors occurring in a nested function** can create a particularly difficult problem. Several **exception-handling** approaches have been attempted to address the problem. One approach, for instance, is to have each function **return an error indication**, either in a **separate variable**, or as a **special range** of values for the normal **return value.**" (*result represents an exception*).

As Per claim 7, Nilsson discloses:

- ***A method according to claim 4, wherein said step of adding new code further includes: adding code that jumps to a subroutine representing a Finally block after invoking said additional method; and adding code that is to be executed after returning from said subroutine.***

For claim 4 feature see claim 4 rejection, for claim 7 feature see Nilsson's column 7, lines 1-7, "the present invention provides a new microprocessor or other data processing system with a new command labeled herein JSRC, standing **for jump to subroutine** (*adding code that jumps to a subroutine)* with context data. The new command has all of the features of the traditional **jump to subroutine commands**, plus an **extra long word that contains an address to the context table**, actual context data, or a combination of both." And fetching logic to process the exception, see description in Nilsson's column 7, lines 24-53 (*Finally block after invoking said additional method and adding code that is to be executed after returning from said subroutine*).

As Per claim 9, Nilsson discloses:

- ***A method according to claim 1, wherein said step of adding new code includes: adding start byte code; adjusting byte code indices; adding exit byte code; and modifying an exception table for said first method.***

For claim 1 feature see claim 1 rejection, for rest of claim 9 feature see Nilsson's column 7, lines 54-61, "the system includes a program store coupled to the instruction fetch logic in which instructions in the set of instructions are stored in locations accessible in response to the instruction pointer. The in-code context data is **stored** following the context-call-instruction **by an offset amount of zero or more bytes**, (*starting byte code and an exception table*) and the logic which updates the instruction pointer jumps over a filed of data at the offset having a length equal to the determinant length"; and column 8, lines 6-11, "In alternative embodiment, the in-code context data comprises a **plurality of bytes of data**, and includes a field for specifying a format for the **plurality of bytes**. Thus for example the **field indicates whether the plurality of bytes** includes immediate context data (*adjusting byte code indices)*, and whether the **plurality of bytes** includes a pointer to context data in another memory location."; see Nilsson's Fig. 8, column 19, lines 6-8, "Next as indicated at block 812, when the handler **exits** the **try block** (*exit byte code*), the **exception stack** is popped and the destructor for the current object to throw is called."

As Per claim 10, Nilsson discloses:

- ***A method according to claim 9, wherein said step of adding exit byte code includes: adding byte code to report said result and jump to a subroutine representing a Finally block; adding byte code to report an exception and jump to said subroutine representing said Finally block; and adding byte code for said subroutine representing said Finally block.***

For claim 9 feature see claim 9 rejection, for jump to subroutine feature see claim 7 rejection, for rest of claim 10 feature see Nilsson's Fig. 7 and Fig. 8, where the unwinding stack process discloses the multiple layers of blocks, and 'try and catch' handler, see descriptions in Nilsson's column 17 into column 19; specifically, see Nilsson's column 17, lines 51-61, "However they are supposed to be implemented by the compiler adding an **implicit try block to be the outermost of the function** *(Finally block)*, that catches any violations, and transforms them into calls to 'unexpectedo', 'terminate( )' or into the effect of throw std::bad_exception".

As Per claim 11, Nilsson discloses:

- ***A method according to claim 1, wherein said step of adding new code includes: adding Try-Finally functionality.***

For claim 1 feature see claim 1 rejection, for validation function of claim 11 feature see Nilsson's column 18, lines 14-20, "For each pair in the object list of the **try block context**, as indicated at block 804 the destructor for such object is called (block 805). When these operations call destructors or copy constructors, those **calls must make sure that they do not let through any exceptions.** (*adding new code for Try-Finally functionality*) This can be accomplished by wrapping the calls in the effects of a "**try [[call]]catch(** …) [terminate( );]" **block**" – Nilsson's try-catch block is the same as the 'Try-Finally', which provides code to access return values and code to access exceptions.

As Per claim 12, Nilsson discloses:

- ***A method for accessing information, comprising: storing a result for a first method from an operand stack; preparing said operand stack for an invocation of a second method; invoking said second method, including providing said result to said second method; and resetting said operand stack with respect to said result to a state existing prior to said step of storing said result.***

Nilsson's disclosure is for accessing stored information and preparing operand stack, see claims 1, 4 rejections.

As Per claim 13, Nilsson discloses:

- ***A method according to claim 12, wherein: said result includes a data item to be returned by said first method.***

For claim 12 feature see claim 12 rejection, for rest of claim 13 feature see claim 2 rejection.

As Per claim 14, Nilsson discloses:

- ***A method according to claim 13, further comprising: returning said result after said step of resetting.***

For claim 13 feature see claim 13 rejection, for rest of claim 14 feature see Nilsson's column 19, lines 14-17, "If the exception stack is not empty, then the

exception object is set to the top of the exception stack (block 816), and the process loops to block 802." – a new result will always be reset when a new exception has occurred.

As Per claim 15, Nilsson discloses:

- ***A method according to claim 12, wherein: said result includes a reference to an exception.***

For claim 12 feature see claim 12 rejection, for rest of claim 15 feature see claim 3 rejection.

As Per claim 16, Nilsson discloses:

- ***A method according to claim 15, further comprising: throwing said exception after said step of resetting.***

For claim 15 feature see claim 15 rejection, for rest of claim 16 feature see claim 5 rejection.

As Per claim 17, Nilsson discloses:

- ***A method according to claim 12, further comprising: performing said second method in response to said step of invoking.***

For claim 12 feature see claim 12 rejection, for rest of claim 17 feature see claim 7 rejection.

As Per claim 18, Nilsson discloses:

- ***A method according to claim 12, further comprising: performing one or more instructions of said first method prior to said step of storing said result, said step of performing one or more instructions includes generating said result.***

For claim 12 feature see claim 12 rejection, for rest of claim 18 feature see claim 1 rejection.

As Per claim 19, Nilsson discloses:

- ***A method according to claim 12, further comprising: returning said result subsequent to said step of resetting; jumping to a subroutine representing a Finally block after invoking said second method and prior to returning said result; and returning from said subroutine prior to returning said result.***

For claim 12 feature see claim 12 rejection, for rest of claim 19 feature see claim 7 rejection.

As Per claim 21, Nilsson discloses:

- ***A method according to claim 12, further comprising: modifying byte code for said first method to add new code that performs said steps of storing, preparing, invoking and resetting.***

For claim 12 feature see claim 12 rejection, for rest of claim 21 feature see claim 9 rejection.

As Per claim 22, Nilsson discloses:

- ***A method according to claim 21, wherein said step of modifying includes: adding start byte code; adjusting byte code indices; adding exit byte code; and modifying an exception table for said first method.***

For claim 21 feature see claim 21 rejection, for rest of claim 22 feature see claim 9 rejection.

As Per claim 23, Nilsson discloses:

- ***A method according to claim 22, wherein said step of adding exit bytes code includes: adding byte code to report said result and jump to a subroutine***

representing a Finally block; adding byte code to report an, exception and jump to said subroutine representing said Finally block; and adding byte code for said subroutine representing said Finally block.

For claim 22 feature see claim 122 rejection, for rest of claim 23 feature see claim 10 rejection.

As Per claim 24, Nilsson discloses:

- ***A method according to claim 21, wherein said step of modifying includes: adding Try-Finally functionality.***

For claim 21 feature see claim 21 rejection, for rest of claim 24 feature see claim 11 rejection.

As Per claim 25, Nilsson discloses:

- ***A method according to claim 12, further comprising: performing said second method, including accessing said result.***

For claim 12 feature see claim 12 rejection, for rest of claim 25 feature see claims 1, 4 rejections, wherein Nilsson's teaching can be used for a second method.

As Per claim 26 , Nilsson discloses:

- ***A method according to claim 12, further comprising: performing said second method, including storing said result for use outside of a thread that includes said first method.***

For claim 12 feature see claim 12 rejection, for rest of claim 26 feature see claim 1 rejection.

As Per claim 29, Nilsson discloses:

- ***One or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor***

readable code for programming one or more processors to perform a method comprising: accessing existing object code, said existing object code includes a first method, said first method is capable of providing a result; and adding new code to said first method, said new code provides said result to said additional method.

Nilsson's teaching also applies for one or more processor readable storage devices having processor readable code; claim 29 is one or more processor readable storage devices' version of claim 1, it is rejected on the same basis as claim 1.

As Per claim 30, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 29, wherein: said result is a data item to be returned by said first method.***

For claim 29 feature see claim 29 rejection, for rest of claim 30 feature see claim 2 rejection.

As Per claim 31, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 29, wherein: said result is a reference to an exception.***

For claim 29 feature see claim 29 rejection, for rest of claim 31 feature see claim 3 rejection.

As Per claim 32, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 29, wherein said step of adding new code includes: adding code that stores said result for said first method from an operand stack; adding code that prepares said operand stack for an invocation of said additional method; adding code that invokes said additional method, including providing said result to said***

additional method; and adding code that resets said operand stack with respect to said result to a state existing prior to storing said result.

For claim 29 feature see claim 29 rejection, for rest of claim 32 feature see claim 4 rejection.

As Per claim 34, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 29, wherein said step of adding new code includes: adding start byte code; adjusting byte code indices; adding exit byte code; and modifying an exception table for said first-method.***

For claim 29 feature see claim 29 rejection, for rest of claim 34 feature see claim 9 rejection.

As Per claim 35, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 34, wherein said step of adding exit byte code includes: adding byte code to report said result and jump to a subroutine representing a Finally block; adding byte code to report an exception and jump to said subroutine representing said Finally block; and adding byte code for said subroutine representing said Finally block.***

For claim 34 feature see claim 34 rejection, for rest of claim 35 feature see claim 10 rejection.

As Per claim 36, Nilsson discloses:

- ***An apparatus that adds functionality in order to access information, comprising: a communication interface; a processor readable storage device; and one or more processors in communication with said processor readable storage device and said communication interface, said one or more processors***

perform a method comprising: access existing object code, said existing object code includes a first method, said first method is capable of providing a result, and adding new code to said first method, said new code provides said result value to said additional method.

Nilsson's teaching also applies for an apparatus, which comprising a communication interface (see Nilsson's Fig. 1) and a processor readable storage device (see Nilsson's Fig. 2), and one or more processors in communication with the communication interface; claim 36 is an apparatus version of claim 1, it is rejected on the same basis as claim 1.

As Per claim 37, Nilsson discloses:

- ***An apparatus according to claim 36, wherein: said result is a data item or a reference to an exception.***

For claim 36 feature see claim 36 rejection, for rest of claim 37 feature see claim 3 rejection.

As Per claim 38, Nilsson discloses:

- ***An apparatus according to claim 36, wherein said step of adding new code includes: adding code that stores said result for said first method from an operand stack; adding code that prepares said operand stack for an invocation of said additional method; adding code that invokes said additional method, including providing said result to said additional method; and adding code that resets said operand stack with respect to said result to a state existing prior to storing said result.***

For claim 38 feature see claim 38 rejection, for rest of claim 38 feature see claim 4 rejection.

As Per claim 41, Nilsson discloses:

- ***An apparatus that adds functionality to existing code in order to access information, comprising: a communication interface; a processor readable storage device; and one or more processors in communication with said processor readable storage device and said communication interface, said one or more processors perform a method comprising: storing a result for a first method from an operand stack, preparing said operand stack for an invocation of a second method, invoking said second method, including providing said result to said second method, and resetting said operand stack with respect to said result to a state existing prior to said step of storing said result.***

Claim 41 is an apparatus version of claim 1, it is rejected on the same basis as claims, 1, 4, 36 rejections.

As Per claim 42, Nilsson discloses:

- ***An apparatus according to claim 41, wherein: said result is a data item to be returned by said first method.***

For claim 41 feature see claim 41 rejection, for rest of claim 42 feature see claim 2 rejection.

As Per claim 43, Nilsson discloses:

- ***An apparatus according to claim 41, wherein: said result is a reference to an exception.***

For claim 41 feature see claim 41 rejection, for rest of claim 43 feature see claim 3 rejection.

As Per claim 45, Nilsson discloses:

- ***An apparatus according to claim 41, wherein said method further comprises: modifying byte code for said first method to add new code that performs said steps of storing, preparing, invoking and resetting.***

For claim 41 feature see claim 41 rejection, for rest of claim 45 feature see claim 9 rejection.

As Per claim 46, Nilsson discloses:

- ***An apparatus according to claim 45, wherein said step of modifying includes the steps of: adding start byte code; adjusting byte code indices; adding exit byte code; and modifying an exception table for said first method.***

For claim 45 feature see claim 45 rejection, for rest of claim 46 feature see claim 9 rejection.

As Per claim 47, Nilsson discloses:

- ***An apparatus according to claim 46, wherein said step of adding exit byte code includes: adding byte code to report said result and jump to a subroutine representing a Finally block; adding byte code to report an exception and jump to said subroutine representing said Finally block; and adding byte code for said subroutine representing said Finally block.***

For claim 46 feature see claim 46 rejection, for rest of the claim 47 feature see claim 10 rejection.

Claim Rejections - 35 USC § 103

11. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> ***(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary***

> ***skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.***

12. Claims 8, 20, 27, 28, 33, 39, 40, and 44 are rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent No. 6,289,446, by Nilsson, hereinafter "Nilsson", in view of well known skills for the people in the art.

As Per claim 8, Nilsson discloses:

- ***A method according to claim 1, wherein: said first method is a Java method.*** For claim 1 feature see claim 1 rejection, Nilsson teaches all aspects of claim 8, but he does not disclose 'a Java method' explicitly, however, using Java is a well known skill t the people in the art, see Nilsson's column 1, lines 30-44, "all other programming languages represent ways of structuring 'human' language so that humans can get computers to perform specific tasks." Further in lines 39-44, "While it is possible for humans to compose meaningful programs in machine code, practically all software development today employs one or more of the available programming languages. The most widely-used programming languages are the 'high-level' languages" It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement Nilsson's disclosure of the method of implementing exception handler by using **JAVA**. The modification would be obvious because one of ordinary skill in the art would be motivated to use a high-level programming language (Nilsson's column 1, lines 43-44).

As Per claim 20, Nilsson discloses:

- ***A method according to claim 12, wherein: said first method is a Java method.*** For claim 12 feature see claim 12 rejection, for rest of claim 20 feature see claim 8 rejection.

As Per claim 27, Nilsson discloses:

- ***A method according to claim 12, further comprising: performing said second method in response to said step of invoking, said second method stores said result; performing one or more instructions of said first method prior to said step of storing said result, said step of performing one or more instructions includes generating said result, said first method is a Java method; and modifying byte code for said first method to add new code that performs said steps of storing, preparing, invoking and resetting.***

For claim 12 feature see claim 12 rejection, for rest of claim 27 feature see claims 1, and 8 rejection.

As Per claim 28, Nilsson discloses:

- ***A method according to claim 27, further comprising: returning said result; jumping to a subroutine representing a Finally block after invoking said second method and prior to returning said result; and returning from said subroutine prior to returning said result.***

For claim 27 feature see claim 27 rejection, for rest of claim 28 feature see claim 10 rejection.

As Per claim 33, Nilsson discloses:

- ***One or more processor readable storage devices according to claim 29, wherein: said first method is a Java method.***

For claim 29 feature see claim 29 rejection, for rest of claim 33 feature see claim 8 rejection.

As Per claim 39, Nilsson discloses:

- *An apparatus according to claim 36, wherein said step of adding new code includes: adding start Java byte code; adjusting Java byte code indices; adding exit Java byte code; and modifying an exception table for said first method.*

For claim 36 feature see claim 36 rejection, for rest of claim 39 feature see claim 8 and claim 9 rejection.

As Per claim 40, Nilsson discloses:

- *An apparatus according to claim 39, wherein said step of adding exit byte code includes: adding byte code to report said result and jump to a subroutine representing a Finally block; adding byte code to report an exception and jump to said subroutine representing said Finally block; and adding byte code for said subroutine representing said Finally block.*

For claim 39 feature see claim 39 rejection, for rest of claim 40 feature see claim 7 rejection.

As Per claim 44, Nilsson discloses:

- *An apparatus according to claim 41, wherein: said first method is a Java method.*

For claim 41 feature see claim 41 rejection, for rest of claim 44 feature see claim 8 rejection.

Conclusion

13. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Berry, US Patent No. 6,728,955, discloses a method for user to specify a vector of while profiling a program. The vector of metrics may optionally be thread-relative. In response to a notification of an occurrence of the current event, a thread-relative elapsed metric is computed by: determining a current thread; retrieving a stored reference metric for the preceding event of the current thread; obtaining a current reference metric; and computing the thread-relative elapsed metric as a difference between the current reference metric and the stored reference metric.

Yellin et al., US Patent No. 6,477,702 , discloses a program interpreter for computer programs written in a bytecode language, which uses a restricted set of data type specific bytecodes. The interpreter, prior to executing any bytecode program, executes a bytecode program verifier procedure that verifies the integrity of a specified program by identifying any bytecode instruction that would process data of the wrong type for such a bytecode and any bytecode instruction sequences in the specified program that would cause underflow or overflow of the operand stack.

Van Dyke et al., US Patent No. 6,934,132, discloses a method for a computer, that has a multi-stage execution pipeline and an instruction decoder. The pipeline exception circuitry is designed to recognize an exception occurring in an instruction after a first side-effect of the instruction has been architecturally committed, and thereupon, to architecturally expose in the processor registers information describing a processor state of the computer, including an intra-instruction program counter value, and to transfer execution to an exception handler. Pipeline resumption circuitry is effective, after completion of the software exception handler, to resume execution of the excepted program based on the information in the processor registers.

Kukol, U.S. Patent No. 5,628,016, discloses a development system having a compiler that allows programmers and software developers to more efficiently develop compiled applications with runtime exception handling support is described. The compiler implements methods for handling of exceptions, which may occur during runtime execution of the program. In an exemplary embodiment, the system of the present invention registers exception handling information (e.g., an Exception

Registration Record) with the underlying operating system, during execution of prolog code for each function (or other discrete section of code).

14. The following summarizes the status of the claims:

35 USC § 112 (2) rejection: Claim 1

35 USC § 101 rejection: Claims 12-17, 25-26

35 USC § 102 rejection: Claims 1-7, 9-19, 21-26, 29-32, 34-38, 41-43, 45-47

35 USC § 103 rejection: Claims 8, 20, 27, 28, 33, 39, 40, and 44

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chih-Ching Chow whose telephone number is 571-272-3693. The examiner can normally be reached on 8:00am - 4:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Zhen can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300. Any inquiry of a general nature of relating to the status of this application should be directed to the **TC2100 Group receptionist: 571-272-2100**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Chih-Ching Chow
Examiner
Art Unit 2191
January 4, 2007

CC

WEI ZHEN
[illegible] PATENT EXAMINER